

On vertex covers and matching number of trapezoid graphs

ALEKSANDAR ILIĆ [‡]

Faculty of Sciences and Mathematics, Višegradska 33, 18000 Niš, Serbia

e-mail: aleksandari@gmail.com

ANDREJA ILIĆ

Faculty of Sciences and Mathematics, Višegradska 33, 18000 Niš, Serbia

e-mail: andrejko.ilic@gmail.com

June 14, 2011

Abstract

The intersection graph of a collection of trapezoids with corner points lying on two parallel lines is called a trapezoid graph. Using binary indexed tree data structure, we improve algorithms for calculating the size and the number of minimum vertex covers (or independent sets), as well as the total number of vertex covers, and reduce the time complexity from $O(n^2)$ to $O(n \log n)$, where n is the number of trapezoids. Furthermore, we present the family of counterexamples for recently proposed algorithm with time complexity $O(n^2)$ for calculating the maximum cardinality matching in trapezoid graphs.

Keywords: trapezoid graphs; vertex cover; matching; algorithms; data structures; binary indexed tree.

AMS Classifications: 05C85, 68R10.

1 Introduction

A trapezoid diagram consists of two horizontal lines and a set of trapezoids with corner points lying on these two lines. A graph $G = (V, E)$ is a trapezoid graph when a trapezoid diagram exists with trapezoid set T , such that each vertex $i \in V$ corresponds to a trapezoid $T(i)$ and an edge exists $(i, j) \in E$ if and only if trapezoids $T(i)$ and $T(j)$ intersect within the trapezoid diagram. A trapezoid $T(i)$ between these lines has four corner points $a(i)$, $b(i)$, $c(i)$ and $d(i)$ – which represent the upper left, upper right, lower left and lower right corner points of trapezoid i , respectively. No two trapezoids share a common endpoint.

Ma and Spinrad [23] showed that trapezoid graphs can be recognized in $O(n^2)$ time, while Mertzijs and Corneil [24] designed structural trapezoid recognition algorithm based on the vertex splitting method in $O(n(m + n))$ time, which is easier for implementation. Here n stands for the number of vertices and m for the number of edges. Trapezoid graphs are perfect, subclass of cocomparability graphs and properly contain both interval graphs and permutation graphs [20].

Trapezoid graphs were first investigated by Corneil and Kamula [9]. These graphs and their generalizations were applied in various fields, including modeling channel routing problems in

[‡]Corresponding author.

VLSI design [11] and identifying the optimal chain of non-overlapping fragments in bioinformatics [1]. Many common graph problems, such as minimum connected dominating sets [29], all-pair shortest paths [26], maximum weighted cliques [3], all cut vertices [16], chromatic number and clique cover [12], all hinge vertices [4], minimum vertex covers [22] in trapezoid graphs, can be solved in polynomial time. For other related problems see [6, 10, 18, 19].

Let $G = (V, E)$ be a simple undirected graph with $|V| = n$ and $|E| = m$. A subset $C \subseteq V$ is called a vertex cover (VC for short) of G if and only if every edge in E has at least one endpoint in C . A vertex cover C is called a minimal vertex cover if and only if no proper subset of C is also a vertex cover. The size of a vertex cover is the number of vertices that it contains. A vertex cover with minimum size is called a minimum vertex cover. Notably, a minimum VC is always a minimal VC, but a minimal VC may not be a minimum VC.

Given a graph G and a fixed parameter k , determining whether G has a vertex cover of at most k vertices is one of the best-known NP complete problems [17]. Many recent studies have focused on developing faster algorithms for the VC problem in special classes of graphs. The number of vertex covers in a graph is important, particularly because this characteristic typically arises in problems related to network reliability [28]. Okamoto et al. [27] proposed $O(n + m)$ time algorithms for counting the numbers of independent sets and maximum independent sets in a chordal graph. Lin et al. [20, 21] considered interval graphs and obtained efficient linear $O(n)$ algorithms for counting the number of VCs, minimal VCs, minimum VCs, and maximum minimal VCs in an interval graph. Recently, Lin and Chen [22] presented $O(n^2)$ algorithms for the same vertex cover properties in a trapezoid graph.

In graph theory, the notions of vertex cover and independent set are dual to each other. A subset $I \subseteq V$ is called an independent set of G if and only if every edge in E is incident on no more than one vertex in I .

Theorem 1.1 ([8]) *A set C is a minimal (minimum) vertex cover of G if and only if $V - C$ is an maximal (maximum) independent set of G .*

A matching in a graph is a set of edges in which no two edges are adjacent. A single edge in a graph is obviously a matching. A maximum matching is a matching with the maximum cardinality and the cardinality of the maximum matching is called a matching number. Currently, the best known algorithm for the constructing maximum matching in general graphs is Edmonds' algorithm [14, 25], which is based on the alternating paths, blossom and shrinking and runs in time $O(|E|\sqrt{|V|})$. For special classes of graphs, such as interval, chordal and permutation graphs, more efficient algorithms were designed (see [2, 5, 7] and references therein). Ghosh and Pal [15] presented an efficient algorithm to find the maximum matching in trapezoid graphs, which turns out to be not correct.

The rest of the paper is organized as follows. In Section 2 we introduce the binary indexed tree data structure. In Section 3 we design $O(n \log n)$ time dynamic programming algorithm for calculating the size of the minimum vertex cover, the number of vertex covers and the number of minimum vertex covers, improving the algorithms from [22]. In Section 4 we present family of counterexamples for $O(n^2)$ algorithm from [15] for finding the maximum matching in trapezoid graphs.

2 Binary indexed data structure

The binary indexed tree (BIT) is an efficient data structure introduced by Fenwick [13] for maintaining the cumulative frequencies. The BIT was first used to support dynamic arithmetic data compression and algorithm coding.

Let A be an array of n elements. The binary indexed tree supports the following basic operations:

- (i) for given value x and index i , add x to the element $A(i)$, $1 \leq i \leq n$;
- (ii) for given interval $[1, i]$, find the sum of values $A(1), A(2), \dots, A(i)$, $1 \leq i \leq n$.

Naive implementation of these operations have complexities $O(1)$ and $O(n)$, respectively. We can achieve better complexity, if we speed up the second operation which will also affect the first operation. Another approach is to maintain all partial sums from $A(1)$ to $A(i)$ for $1 \leq i \leq n$. This way the operations have complexities $O(n)$ and $O(1)$, respectively.

The main idea of binary indexed tree structure is that sum of elements from the segment $[1, i]$ can be represented as sum of appropriate set of subsegments. The BIT structure is based on decomposition of the cumulative sums into segments and the operations to access this data structure are based on the binary representation of the index. This way the time complexity for both operations will be the same $O(\log n)$. We want to construct these subsegments such that each element is contained in at most $\log n$ subsegments, which means that we have to change the partial sums of at most $\log n$ subsegments for the first procedure. On the other hand, we want to construct these subsegments such that for every $1 \leq i \leq n$ the subsegment $[1, i]$ is divided in at most $\log n$ subsegments. It follows that these subsegments can be $[i - 2^{r(i)} + 1, i]$, where $r(i)$ is the position of the last digit 1 (from left to right) in the binary representation of the index i . We store the sums of subsegments in the array $Tree$ (see Algorithm 1 and Algorithm 2), and the element $Tree(i)$ represents the sum of elements from index $i - 2^{r(i)} + 1$ to i . The structure is space-efficient in the sense that it needs the same amount of storage as just a simple array of n elements. Instead of sum, one can use any distributive function (such as maximum, minimum, product). We have the following

Proposition 2.1 *Calculating the sum of the elements from $A(1)$ to $A(i)$, $1 \leq i \leq n$, and updating the element $A(i)$ in the binary indexed tree is performed in $O(\log n)$ time.*

The fundamental operation involves calculating a new index by stripping the least significant 1 from the old index, and repeating this operation until the index is zero. For example, to read the cumulative sum $A(1) + A(2) + \dots + A(11)$, we form a sum

$$Tree(11) + Tree(10) + Tree(8) = (A(11)) + (A(9) + A(10)) + (A(1) + A(2) + \dots + A(8)).$$

An illustrative example is presented in Table 1.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	1	0	2	1	1	3	0	4	2	5	2	2	3	1	0	2
$Tree$	1	1	2	4	1	4	0	12	2	7	2	11	3	4	0	29
Cumulative sums	1	1	3	4	5	8	8	12	14	19	21	23	26	27	27	29

Table 1: An example with array A of length $n = 16$, binary indexed tree $Tree$ and cumulative sums.

Using the fast bitwise AND operator, we can calculate the largest power of 2 dividing the integer n simply as $r(n) = n \text{ AND } (-n)$.

Algorithm 1: Updating the binary indexed tree.

Input: The value $value$ and element index $index$.
while $index \leq n$ **do**
 $Tree[index] = Tree[index] + value$;
 $index = index + (index \text{ and } (-index))$;
end

Algorithm 2: Calculating the cumulative sum.

Input: The index $index$.
Output: The sum of elements $A[1], A[2], \dots, A[index]$.
 $sum = 0$;
while $index > 0$ **do**
 $sum = sum + Tree[index]$;
 $index = index - (index \text{ and } (-index))$;
end
return sum ;

3 The algorithm for minimum vertex covers

Let $T = \{1, 2, \dots, n\}$ denote the set of trapezoids in the trapezoid graph $G = (V, E)$. For simplicity, the trapezoid in T that corresponds to vertex i in V is called trapezoid $T(i)$. Without loss of generality, the points on each horizontal line of the trapezoid diagram are labeled with distinct integers between 1 and $2n$.

For simplicity and easier implementation, we will consider the maximum independent sets (according to Theorem 1.1). At the beginning, we add two dummy trapezoids 0 and $n + 1$ to T , where $a(0) = b(0) = c(0) = d(0) = 0$ for the vertex 0 and $a(n + 1) = b(n + 1) = c(n + 1) = d(n + 1) = 2n + 1$ for the vertex $n + 1$. Trapezoid i lies entirely to the left of trapezoid j , denoted by $i \ll j$, if $b(i) < a(j)$ and $d(i) < c(j)$. It follows that \ll is a partial order over the trapezoid set T and (T, \ll) is a strictly partially ordered set.

In order to get $O(n \log n)$ algorithms, we will use binary indexed tree data structure for sum and maximum functions.

3.1 The size of maximum independent set and the total number of independent sets

Arrange all trapezoids by the upper left corner $a(i)$. Let $max_ind(i)$ denote the size of the maximum independent set of $T(1), T(2), \dots, T(i - 1), T(i)$ that contain trapezoid $T(i)$. For the starting value, we set $max_ind(0) = 0$. The following recurrent relation holds

$$max_ind(i) = \max\{1 + max_ind(k), 0 \leq k < i \text{ and } T(k) \ll T(i)\}.$$

This produces a simple $O(n^2)$ dynamic programming algorithm (similar to [22]).

To speed up the above algorithm, we will store the values from max_ind in the binary indexed tree cum_max for maintaining the partial maximums. First initialize each element of cum_max with -1 . We need an additional array $index$, such that $index(j)$ contains the index of the trapezoid with left or right coordinate equal to j . We are going to traverse the coordinates from 0 to $2n + 1$, and let j be the current coordinate. In order to detect which already processed

trapezoids are to the left of the current trapezoid $T(i)$, we calculate the value $max_ind(i)$ when the current coordinate is the left upper coordinate of $a(i)$ and insert it in the binary indexed tree when it is the right upper coordinate $b(i)$. Then we have two cases:

- (i) coordinate j is the upper left coordinate of the trapezoid $T(i)$. We only need to consider the trapezoids that have lower right coordinate smaller than $c(i)$. As explained, these trapezoids have their corresponding values stored in the segment $[1, c(i)]$ in the binary indexed tree and therefore we can calculate $max_ind(i)$ based on the maximum values among $cum_max(1), cum_max(2), \dots, cum_max(c(i))$.
- (ii) coordinate j is the upper right coordinate of the trapezoid $T(i)$. In this case, the value $max_ind(i)$ is already calculated and we put this value in the binary indexed tree at the position $d(i)$.

The final solution is $max_ind(n + 1) - 1$. The pseudo-code is presented in Algorithm 3.

Algorithm 3: The size of the maximum independent set.

Input: The trapezoids T and the array $index$.

Output: The number of maximum independent sets stored in the array max_ind .

Initialize binary indexed tree for maximum cum_max ;

for $j = 0$ **to** $2n + 1$ **do**

$i = index[j]$;

if $a[i] = j$ **then**

$max_ind[i] = \text{Calculate}(c[i]) + 1$;

end

else // $b[i] = j$

Update $(d[i], max_ind[i])$;

end

end

return $max_ind[n + 1] - 1$;

The set of all VCs and the number of VCs of G are denoted by $VC(G)$ and $|VC(G)|$, respectively. Let $N_G(v)$ be the set of vertices adjacent to v in the graph G . In [22] the authors proved the following result.

Lemma 3.1 *For a graph G and arbitrary vertex $v \in V$, it holds*

$$|VC(G)| = |VC(G - v)| + |VC(G - v - N_G(v))|.$$

For counting the total number of independent sets, we can use the same method as above. The only difference is that we will have sum instead of maximum function in the binary indexed tree. Let $num_ind(i)$ denote the number of independent sets of trapezoids $T(1), T(2), \dots, T(i - 1), T(i)$ that contain trapezoid $T(i)$. According to Lemma 3.1 the following recurrent relation holds

$$num_ind(i) = 1 + \sum_{0 \leq k \leq i, T(k) \ll T(i)} num_ind(k).$$

The total number of independent sets is simply $num_ind(n + 1) - 1$ and time complexity is the same $O(n \log n)$.

3.2 The number of maximum independent sets

For counting the number of maximum independent sets, we will need one additional array num_max_ind , such that $num_max_ind(i)$ represents the number of maximum independent sets among trapezoids $T(1), T(2), \dots, T(i-1), T(i)$ such that $T(i)$ belongs to the independent set. After running the algorithm for calculating the array max_ind , for each $1 \leq i \leq n$ we need to sum the number of independent sets of all indices j , such that $T(j) \ll T(i)$ and $max_ind(j) + 1 = max_ind(i)$ in order to get the number of independent sets with maximum cardinality. The pseudo-code of the algorithm for counting the number of maximum independent sets from [22] is presented in Algorithm 4.

Algorithm 4: The number of maximum independent sets.

Input: The trapezoids T and the array max_ind .

Output: The number of maximum independent sets stored in the array num_max_ind .

```

 $num\_max\_ind[0] = 1;$ 
for  $i = 1$  to  $n + 1$  do
     $num\_max\_ind[i] = 0;$ 
    for  $j = 0$  to  $i - 1$  do
        if  $(max\_ind[j] + 1 = max\_ind[i])$  and  $(a[j] > b[i])$  and  $(c[j] > d[i])$  then
             $num\_max\_ind[i] = num\_max\_ind[i] + num\_max\_ind[j];$ 
        end
    end
end
return  $num\_max\_ind[n + 1] - 1;$ 

```

We will use again binary indexed trees for designing $O(n \log n)$ algorithm. The main problem is how to manipulate the partial sums. Namely, for each trapezoid $T(i)$, we need to sum the values $num_max_ind(j)$ of those trapezoids $T(j)$ which are entirely on the left of $T(i)$ with additional condition $max_ind(j) + 1 = max_ind(i)$.

This can be done by considering all pairs $(k, k + 1)$ of two neighboring values $1 \leq k \leq max_ind(n + 1)$. Using two additional arrays for the coordinates and trapezoid indices, we can traverse trapezoids $T(i)$ from left to right, such that $max_ind(i) = k$ or $max_ind(i) = k + 1$. First, for each $1 \leq k \leq max_ind(n + 1)$ one needs to carefully preprocess all trapezoids and store the indices and coordinates of all trapezoids $T(i)$ with $max_ind(i) = k$ in order to keep the memory limit $O(n)$. Therefore, all trapezoids will be stored in the dynamic array of arrays S , such that $S(k)$ contains all trapezoids $T(i)$ with $max_ind(i) = k$. Then we fill the binary indexed tree with the values for trapezoids with $max_ind(i) = k$, and calculate the value $num_max_ind(i)$ for trapezoids with $max_ind(i) = k + 1$, as before. Instead of resetting the binary indexed tree for each k , we can again traverse the trapezoids with $max_ind(i) = k$ and update the values of BIT to 0.

Since every trapezoid will be added and removed exactly once from the binary indexed tree data structure, the total time complexity is $O(n \log n)$. This reusing of the data structure is novel approach to the best of our knowledge and makes this problem very interesting.

We conclude this section by summing the results in the following theorem.

Theorem 3.2 *The proposed algorithms calculate the size and the number of maximum independent sets, as well as the total number of independent sets in time $O(n \log n)$ of trapezoid graph G with n vertices.*

One can easily extend this algorithm for efficient construction the independence polynomial of a trapezoid graph

$$I(G, x) = \sum_{k=0}^{\alpha(G)} s_k x^k,$$

where s_k is the number of independent sets of cardinality k and $\alpha(G)$ is the independence number of G .

4 The algorithm for the maximum matching

Ghosh and Pal [15] presented an efficient algorithm for finding the maximum matching in trapezoid graphs. The proposed algorithm takes $O(n^2)$ time and $O(n + m)$ space. They defined the right spread $f(i)$ of a trapezoid $T(i)$ as the maximum $\max\{b(i), d(i)\}$. Let M be the set of edges which form a matching of the graph G . Deletion of a vertex from the graph means deletion of that vertex and its adjacent edges.

The algorithm is designed as follows. Calculate the right spread $f(i)$ and arrange all trapezoids by f in ascending order for $1 \leq i \leq n$ in linear time $O(n)$. Select the trapezoid i with the minimum value of the right spread $f(i)$. After that, select the second minimum value $f(j)$. If trapezoids $T(i)$ and $T(j)$ are adjacent, then put the edge (i, j) in the maximum matching M , remove the vertices i and j from the graph G and mark the corresponding elements of the array f . If $(i, j) \notin E(G)$ then continue selecting the next minimum elements $f(j')$ from the array f until the trapezoids i and j' are adjacent. In that case remove the vertices i and j' , put the edge (i, j') in the maximum matching M and mark the corresponding elements of f ; otherwise remove the trapezoid i from the graph G since it cannot be matched. Continue this procedure as long as $|V(G)| > 1$. Since for each trapezoid we potentially traverse all trapezoids with larger values $f(i)$, the time complexity of the proposed algorithm is $O(n^2)$.

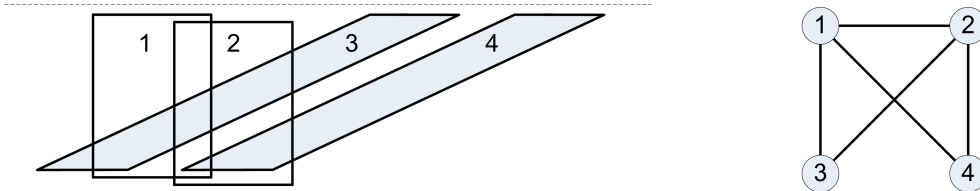


Figure 1: The minimal counterexample for the matching algorithm.

Obviously, the algorithm works for any set of $n = 3$ trapezoids. In the configuration on Figure 2, there are four trapezoids and only trapezoids $T(3)$ and $T(4)$ are not adjacent. The trapezoids are ordered such that $f(1) < f(2) < f(3) < f(4)$. By proposed algorithm in [15], we will match the trapezoids 1 and 2 and get the maximal matching of cardinality 1. But obviously, the size of the maximum matching is 2 by pairing the trapezoids (1, 3) and (2, 4). Therefore, this algorithm is not correct and gives only an approximation for the maximum matching. One can easily construct a family of counterexamples based on this minimal example by adding trapezoids to the left and right.

We leave as an open problem to design more efficient algorithm for finding maximum matching in trapezoid graphs.

Acknowledgement. This work was supported by Research Grants 174010 and 174033 of Serbian Ministry of Education and Science.

References

- [1] M. I. Abouelhoda, E. Ohlebusch, *Chaining algorithms for multiple genome comparison*, J. Discrete Algorithms 3 (2005) 321–341.
- [2] M. G. Andrews, M. J. Atallah, D. Z. Chen, D. T. Lee, *Parallel Algorithms for Maximum Matching in Complements of Interval Graphs and Related Problems*, Algorithmica 26 (2000) 263–289.
- [3] D. Bera, M. Pal, T. K. Pal, *An efficient algorithm to generate all maximal cliques on trapezoid graphs*, Int. J. Comput. Math. 79 (2002) 1057–1065.
- [4] D. Bera, M. Pal, T. K. Pal, *An Efficient Algorithm for Finding All Hinge Vertices on Trapezoid Graphs*, Theory Comput. Systems 36 (2003) 17–27.
- [5] M. S. Chang, *Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs*, Lecture Notes in Computer Science 1178, Springer, Berlin, 1996, 146–155.
- [6] F. Cheah, D. G. Corneil, *On the structure of trapezoid graphs*, Discrete Appl. Math. 66 (1996) 109–133.
- [7] Y. Chung, K. Park, Y. Cho, *Parallel Maximum Matching Algorithms in Interval Graphs*, International Conference on Parallel and Distributed Systems, 1997, 602–609.
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press, 2001.
- [9] D. G. Corneil, P. A. Kamula, *Extensions of permutation and interval graphs*, In: Proceedings of 18th Southeastern Conference on Combinatorics, Graph Theory and Computing, 1987, 267–275.
- [10] C. Crespelle, P. Gambette, *Unrestricted and complete Breadth First Search of trapezoid graphs in $O(n)$ time*, Inform. Process. Lett. 110 (2010) 497–502.
- [11] I. Dagan, M. C. Golumbic, R. Y. Pinter, *Trapezoid graphs and their coloring*, Discrete Appl. Math. 21 (1988) 35–46.
- [12] S. Felsner, R. Müller, L. Wernisch, *Trapezoid graphs and generalizations, geometry and algorithms*, Discrete Appl. Math. 74 (1997) 13–32.
- [13] P. M. Fenwick, *A new data structure for cumulative frequency tables*, Software - Practice and Experience 24 (1994), 327–336.
- [14] H. N. Gabow, *Data structures for weighted matching and nearest common ancestors with linking*, in: Proceedings of the First Annual ACM SIAM Symposium on Discrete Algorithms, 1990, 434–443.
- [15] P. K. Ghosh, M. Pal, *An efficient algorithm to find the maximum matching on trapezoid graphs*, J. Korean Soc. Ind. Appl. Math. IT Series 9 (2) (2005) 13–20.
- [16] M. Hota, M. Pal, T. K. Pal, *Optimal sequential and parallel algorithms to compute all cut vertices on trapezoid graphs*, Comput. Optim. Appl. 27 (1) (2004) 95–113.

- [17] R. M. Karp, *Reducibility among combinatorial problems*, In: R. E. Miller, J. W. Thatcher (Eds.), *Complexity of Computer Computation*, Plenum Press, New York, 1972, 85–103.
- [18] Y. D. Liang, *Domination in trapezoid graphs*, Inform. Process. Lett. 52 (1994) 309–315.
- [19] Y. D. Liang, *Steiner set and connected domination in trapezoid graphs*, Inform. Process. Lett. 56 (1995) 101–108.
- [20] M. S. Lin, *Fast and simple algorithms to count the number of vertex covers in an interval graph*, Inform. Process. Lett. 102 (2007) 143–146.
- [21] M. S. Lin, Y. J. Chen, *Linear time algorithms for counting the number of minimal vertex covers with minimum/maximum size in an interval graph*, Inform. Process. Lett. 107 (2008) 257–264.
- [22] M. S. Lin, Y. J. Chen, *Counting the number of vertex covers in trapezoid graph*, Inform. Process. Lett. 109 (2009) 1187–1192.
- [23] T. H. Ma, J. P. Spinrad, *On the 2-chain subgraph cover and related problems*, J. Algorithms 17 (1994) 251–268.
- [24] G. B. Mertzios, D. G. Corneil, *Vertex splitting and the recognition of trapezoid graphs*, Technical Report AIB-2009-16, RWTH Aachen University, 2009.
- [25] S. Micali, V. V. Vazirani, *An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs*, Proceedings of the 21st IEEE Symposiums on Foundations of Computer Science, 1980, 17–27.
- [26] S. Mondal, M. Pal, T. K. Pal, *An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs*, Int. J. Comput. Eng. Sci. (IJCES) 3 (2002) 103–116.
- [27] Y. Okamoto, T. Uno, R. Uehara, *Counting the number of independent sets in chordal graphs*, J. Discrete Algorithms 6 (2) (2005) 229–242.
- [28] J. S. Provan, M. O. Ball, *The complexity of counting cuts and of computing the probability that a graph is connected*, SIAM J. Computing 12 (4) (1983) 777–788.
- [29] Y. T. Tsai, Y. L. Lin, F. R. Hsu, *Efficient algorithms for the minimum connected domination on trapezoid graphs*, Inform. Sci. 177 (2007) 2405–2417.